In [15]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [3]:

```
df=pd.read_csv('heart (2).csv')
```

In [4]:

```
df
```

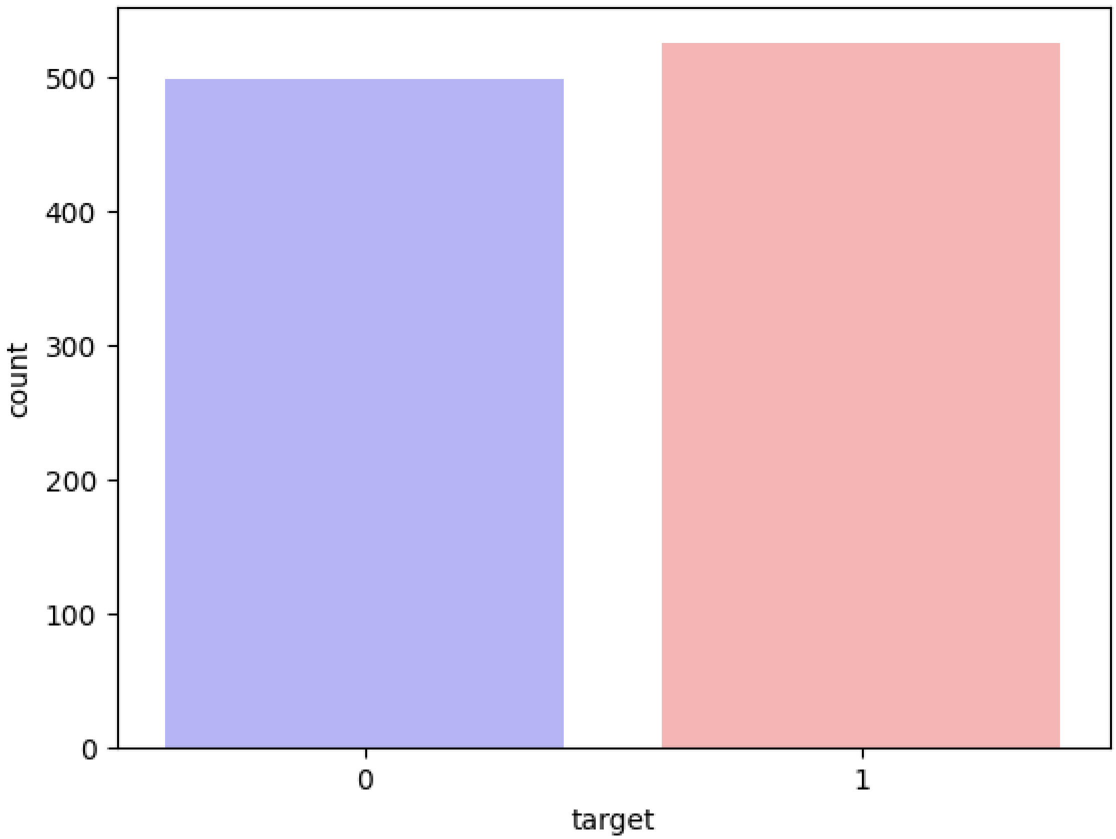
Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

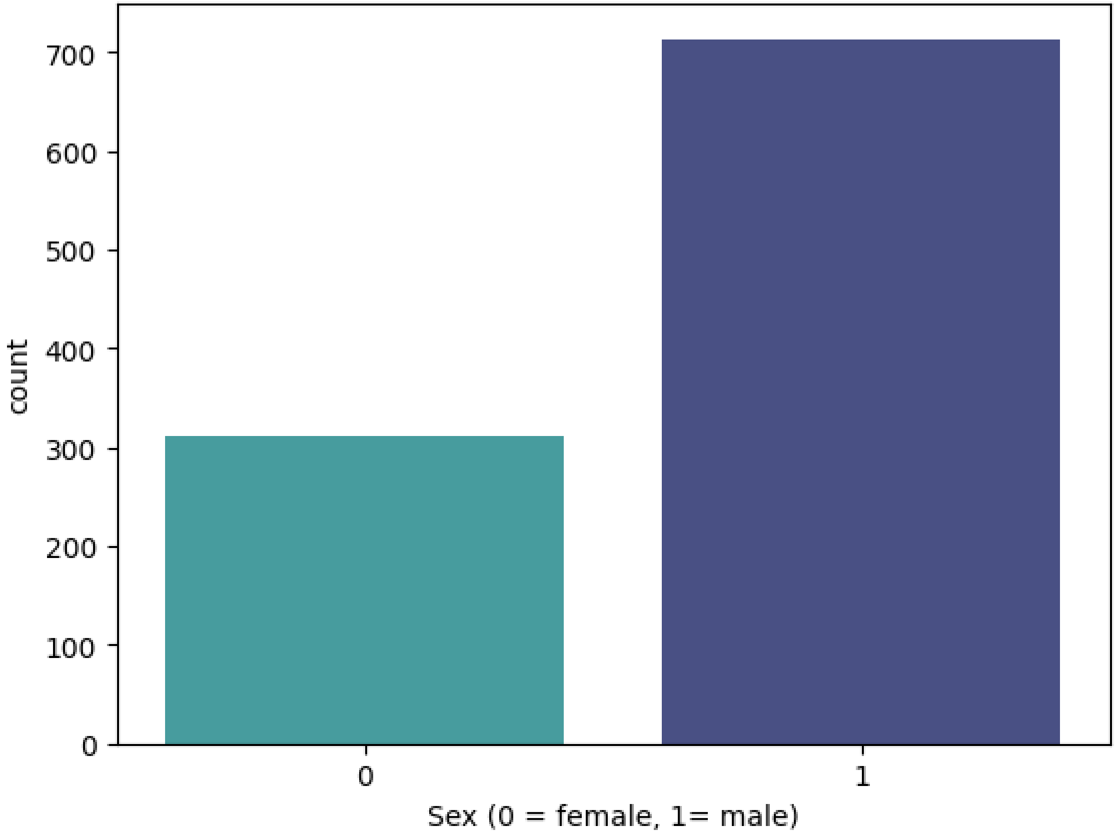
In [5]:

```
sns.countplot(x="target", data=df, palette="bwr")
plt.show()
```



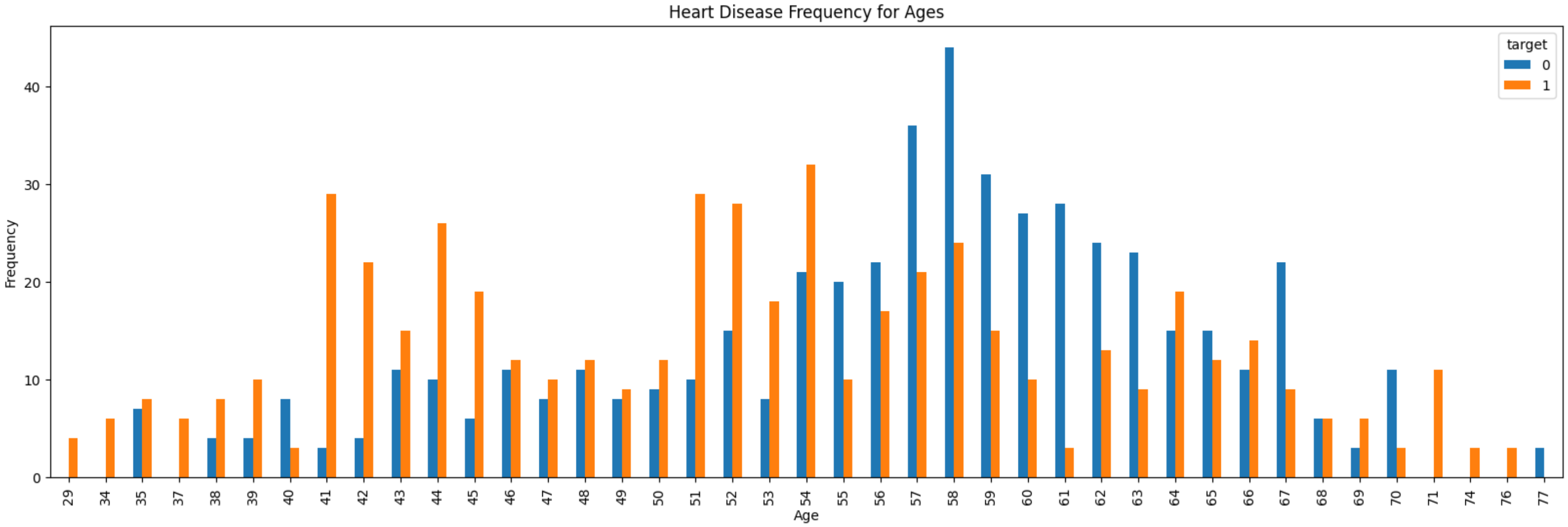
In [6]:

```
sns.countplot(x='sex', data=df, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```



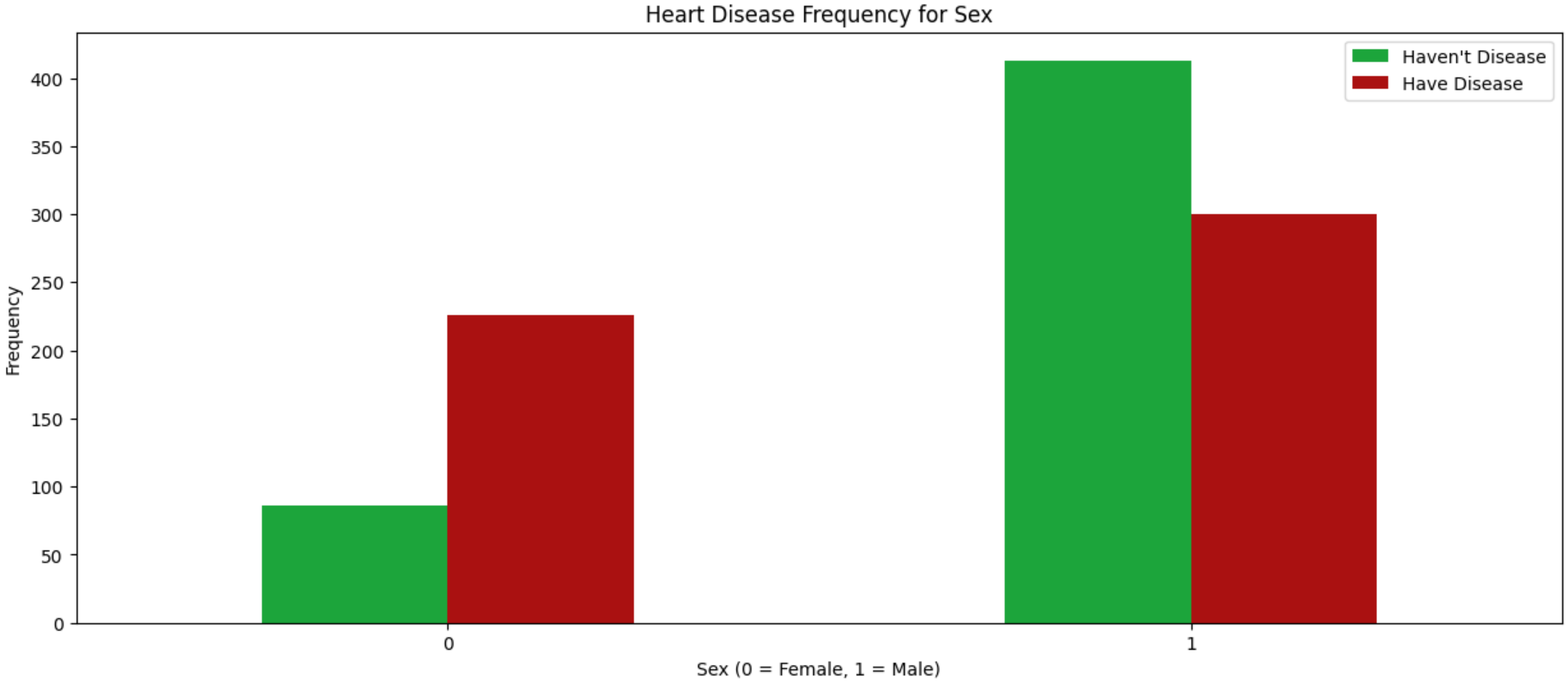
In [7]:

```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```



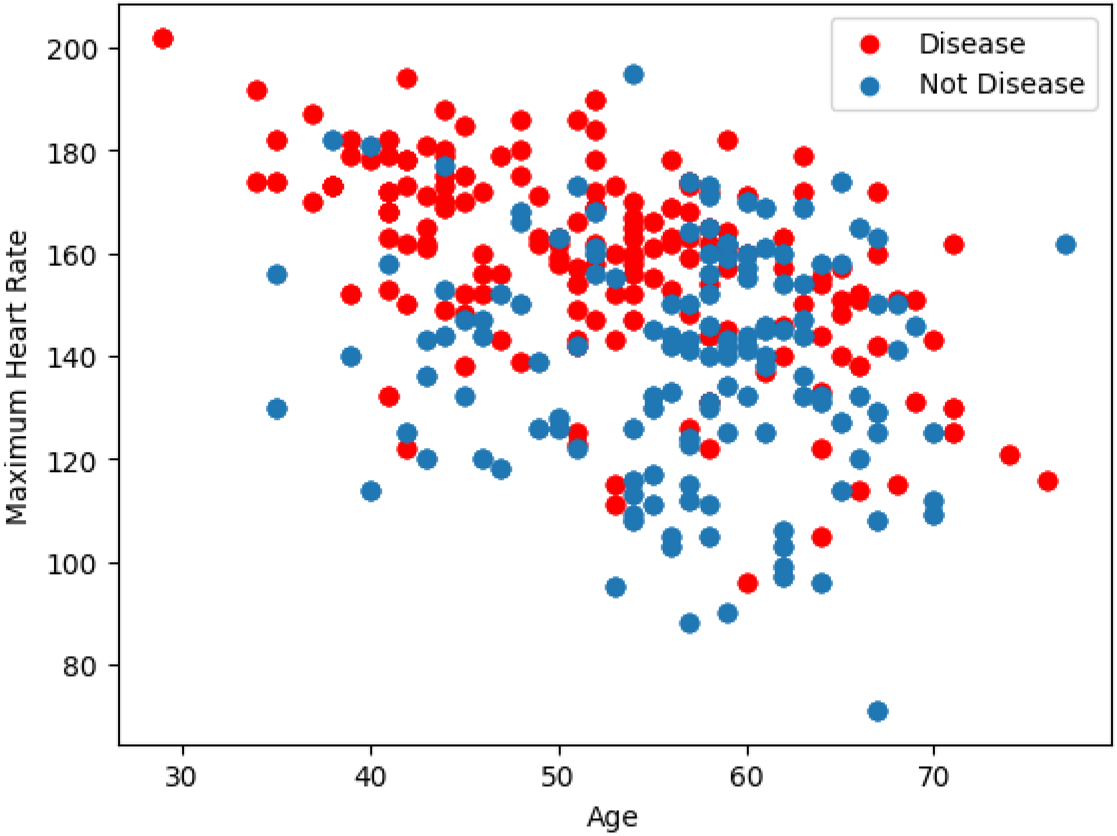
In [8]:

```
pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['#1CA53B','#AA1111' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency')
plt.show()
```



In [9]:

```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```

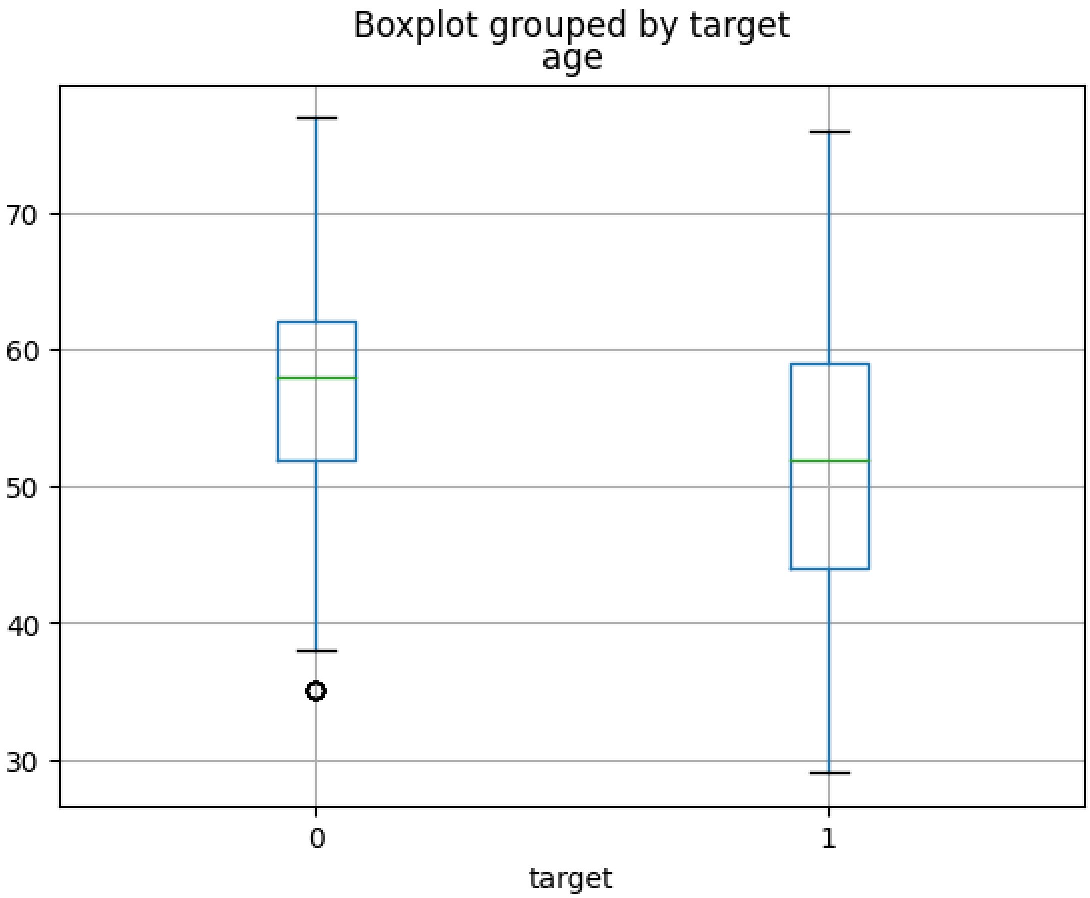


In [25]:

```
df.boxplot(column='age', by = 'target')
```

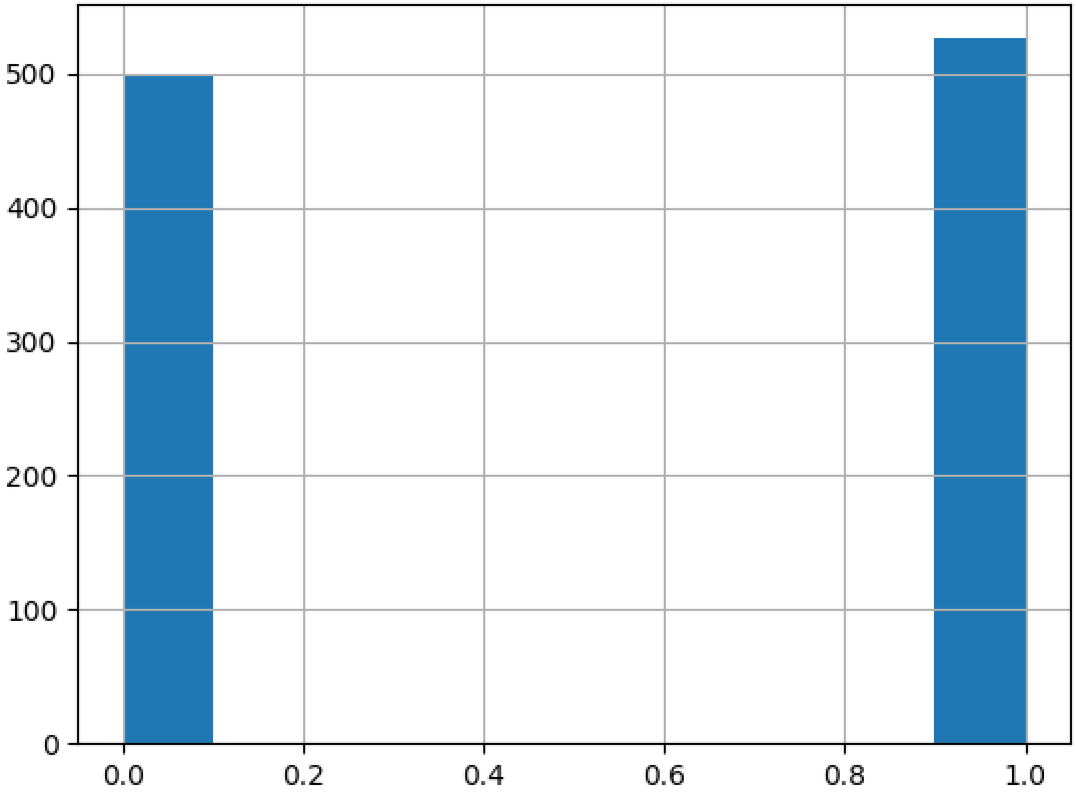
Out[25]:

```
<AxesSubplot:title={'center':'age'}, xlabel='target'>
```



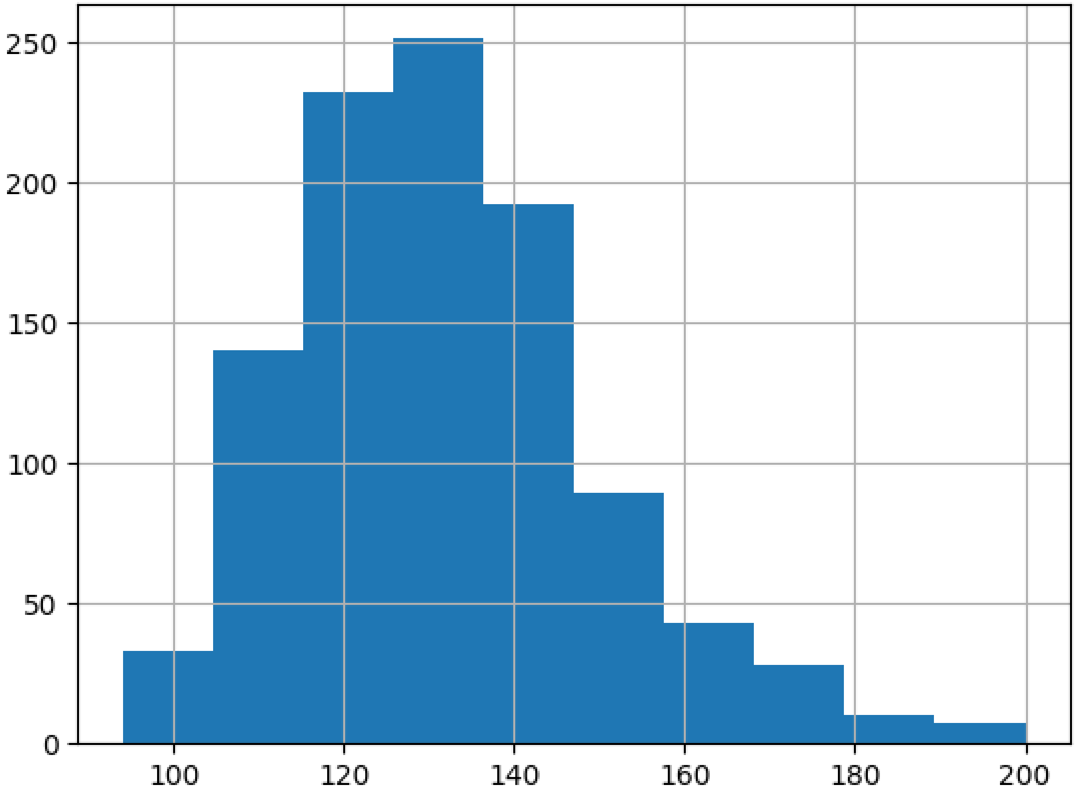
```
In [26]:  df['target'].hist()

Out[26]: <AxesSubplot:>
```



In [27]:
```
df['trestbps'].hist()
```

Out[27]:
```
<AxesSubplot:>
```

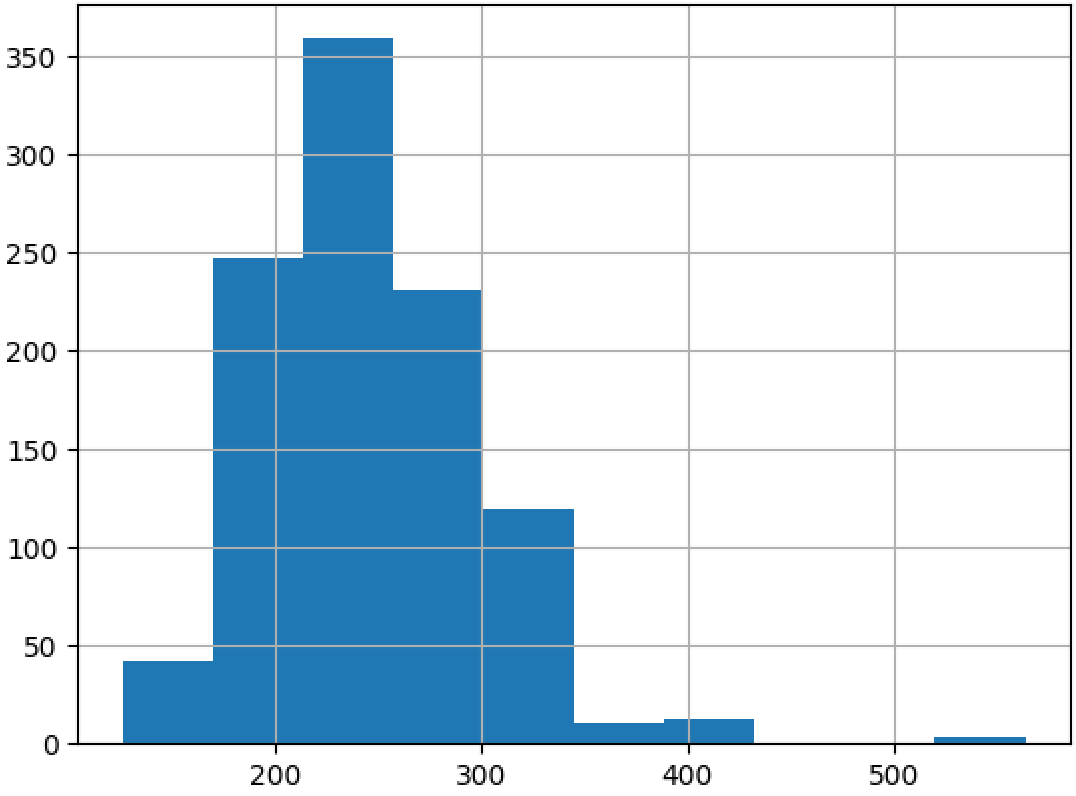


In [28]:

```
df['chol'].hist()
```

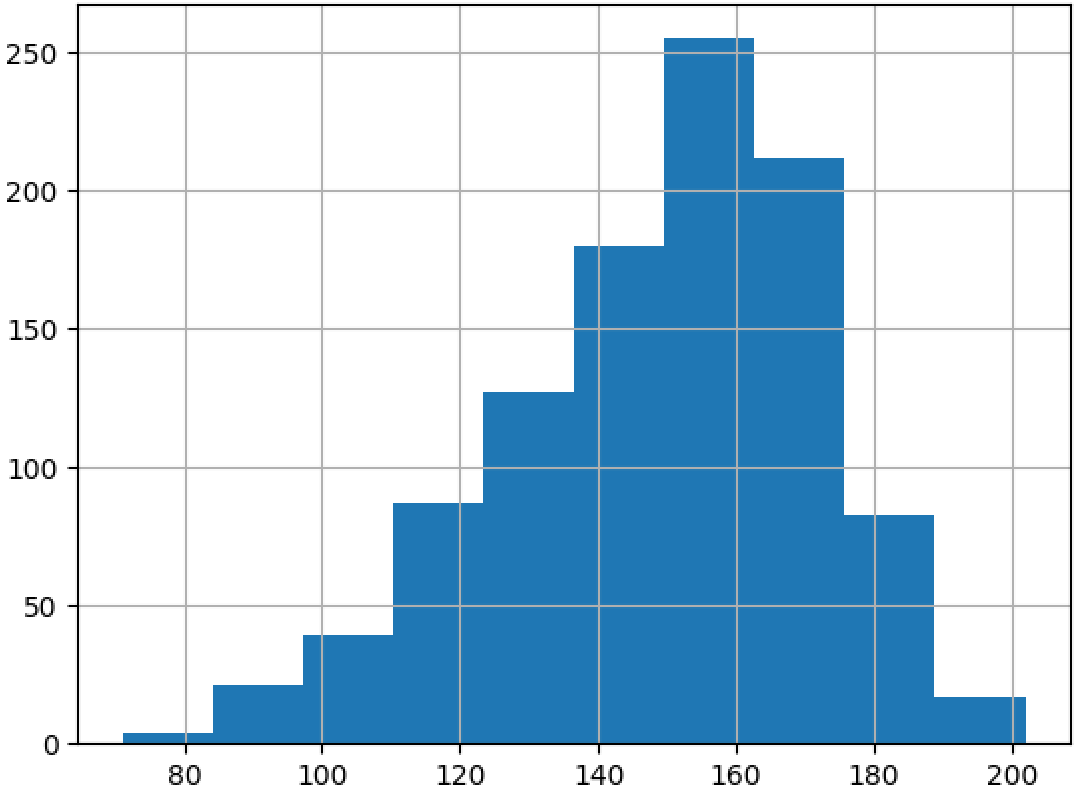
Out[28]:

```
<AxesSubplot:>
```



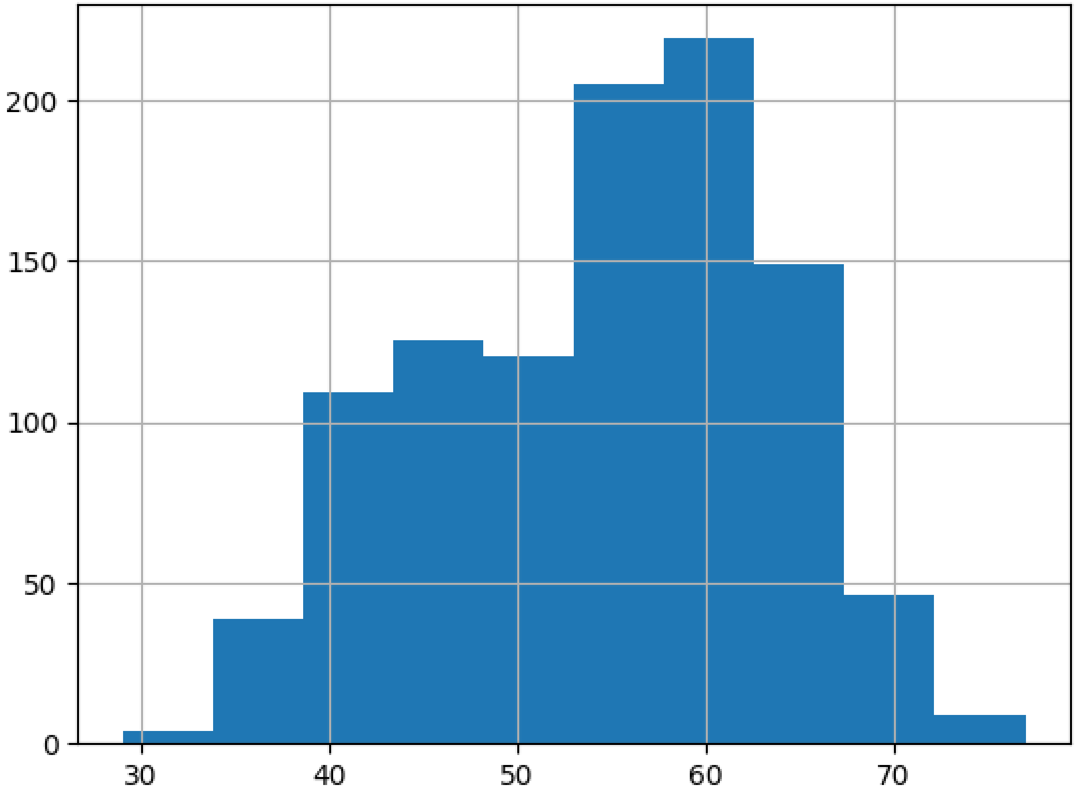
```
In [29]: df['thalach'].hist()
```

```
Out[29]: <AxesSubplot:>
```



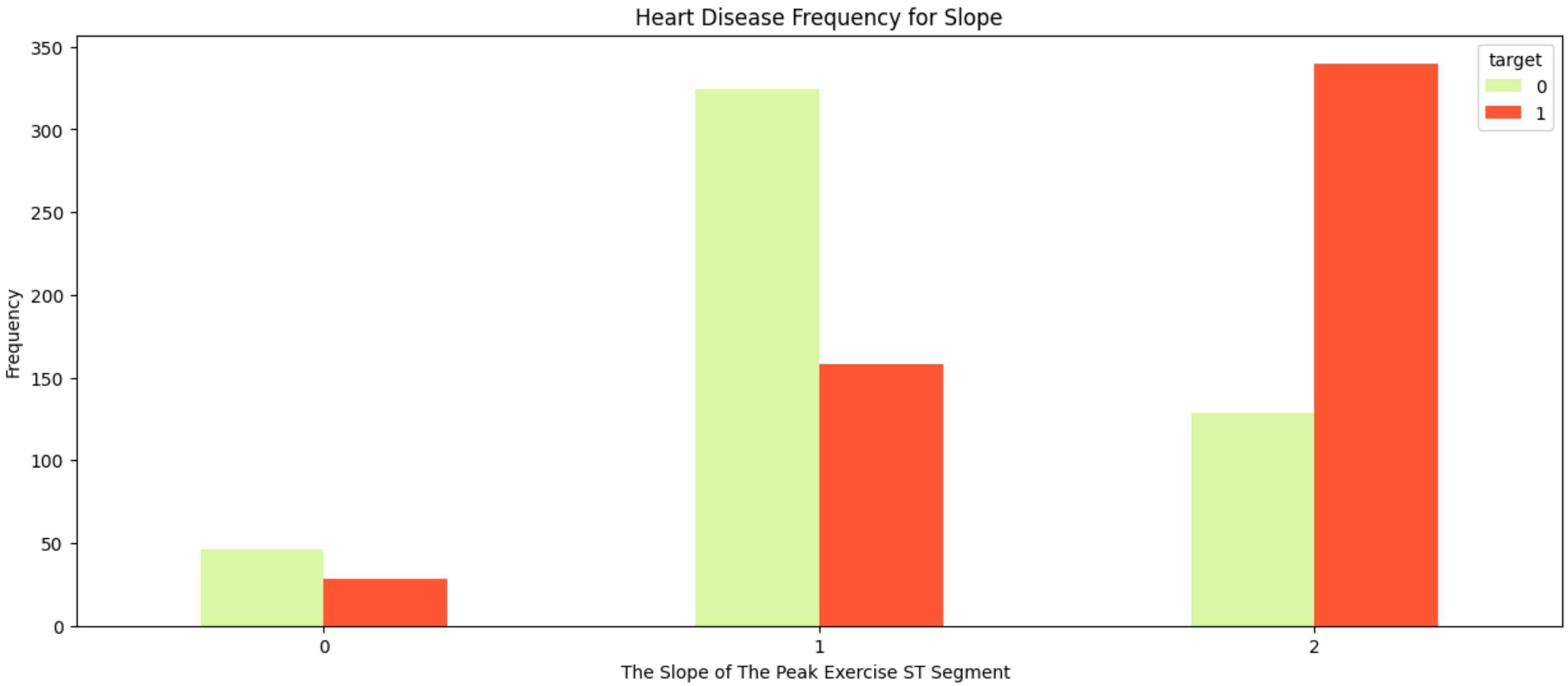
In [30]:
```
df['age'].hist()
```

Out[30]:
```
<AxesSubplot:>
```



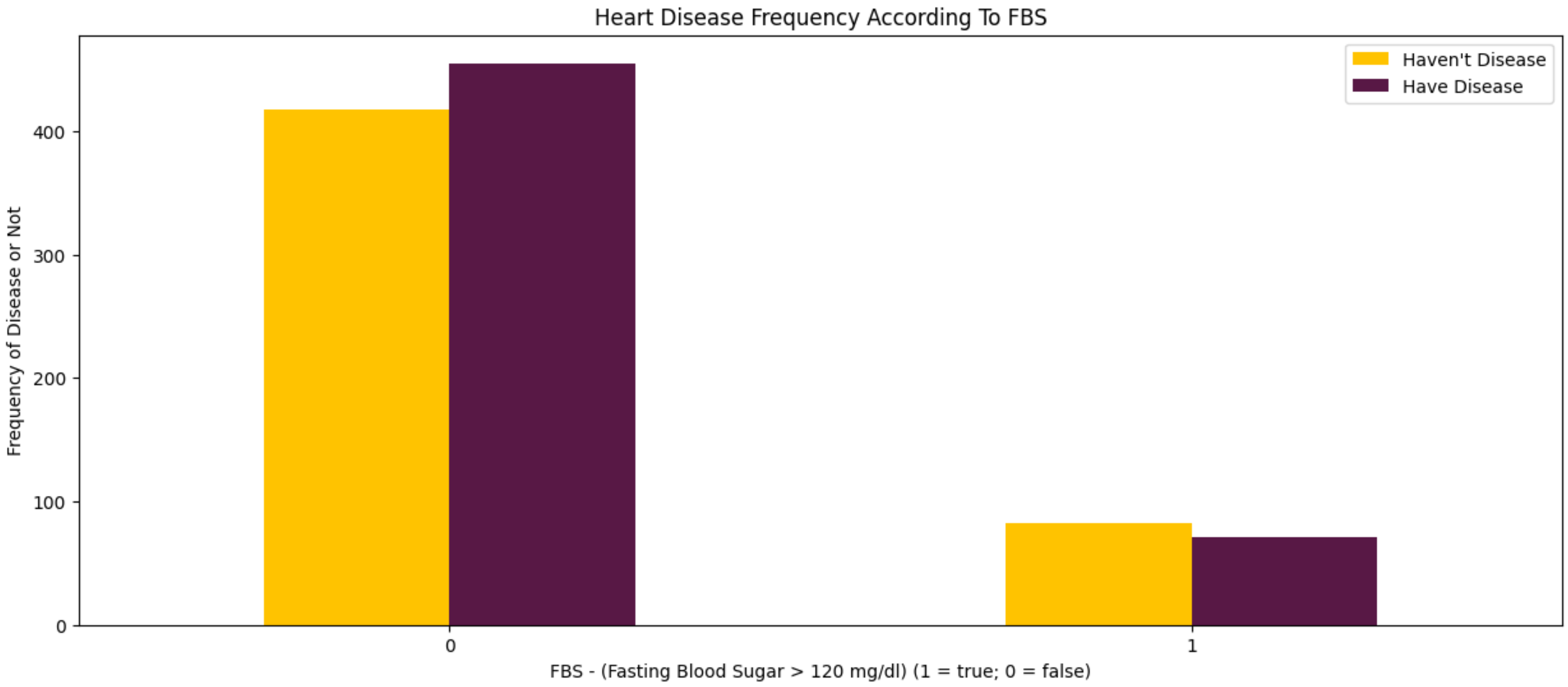
In [10]:

```
pd.crosstab(df.slope,df.target).plot(kind="bar",figsize=(15,6),color=['#DAF7A6','#FF5733' ])
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



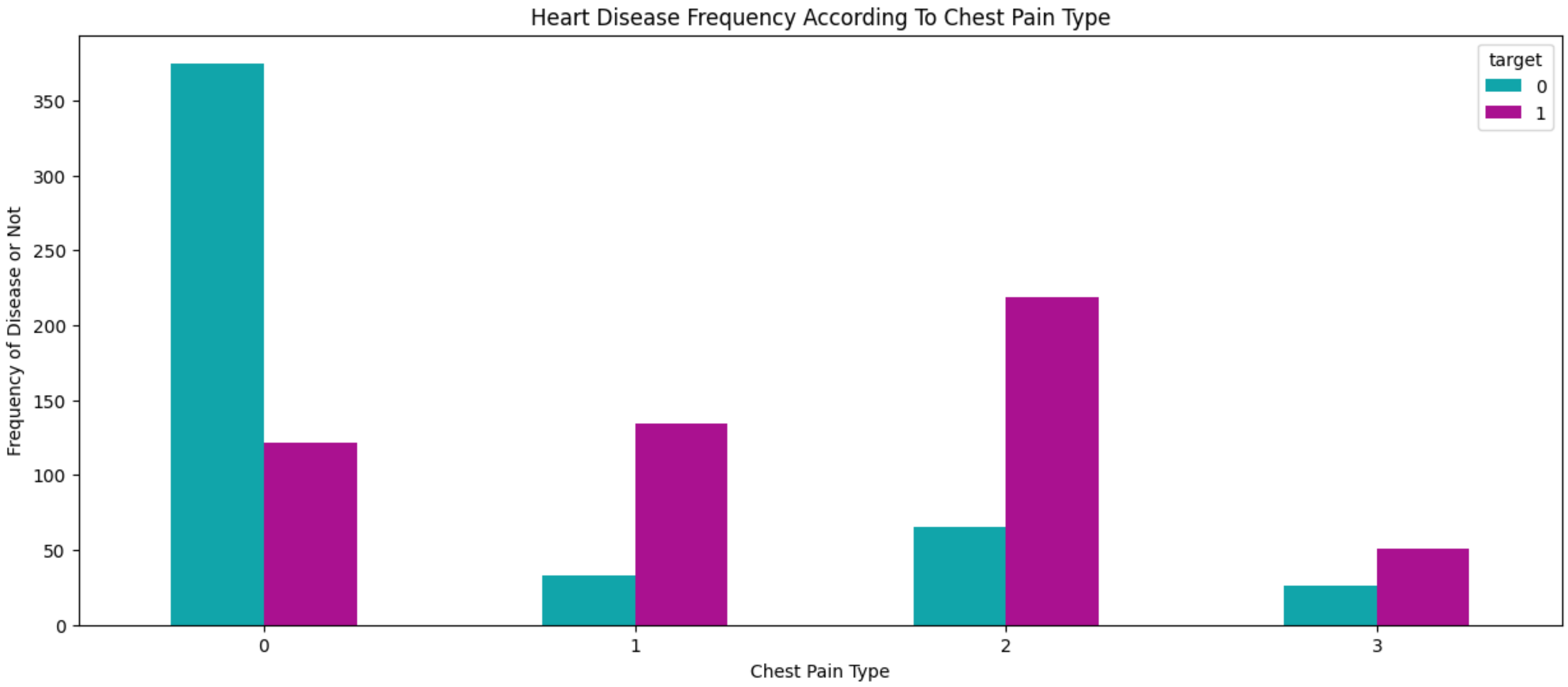
In [11]:

```
pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(15,6),color=['#FFC300','#581845' ])
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



In [12]:

```
pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#11A5AA','#AA1190' ])
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



In [13]:

```
y = df.target.values
x_data = df.drop(['target'], axis = 1)
```

In [16]:

```
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values
```

```
c:\python 39\lib\site-packages\numpy\core\fromnumeric.py:84: FutureWarning: In a future version, Dat
aFrame.min(axis=None) will return a scalar min over the entire DataFrame. To retain the old behavio
r, use 'frame.min(axis=0)' or just 'frame.min()'
  return reduction(axis=axis, out=out, **passkwargs)
c:\python 39\lib\site-packages\numpy\core\fromnumeric.py:84: FutureWarning: In a future version, Dat
aFrame.max(axis=None) will return a scalar max over the entire DataFrame. To retain the old behavio
r, use 'frame.max(axis=0)' or just 'frame.max()'
  return reduction(axis=axis, out=out, **passkwargs)
```

In [17]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=0)
```

In [19]:

```
from sklearn.metrics import accuracy_score
clf = LogisticRegression()

# Train the model on the training set
clf.fit(x_train, y_train)

# Use the trained model to make predictions on the testing set
y_pred = clf.predict(x_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

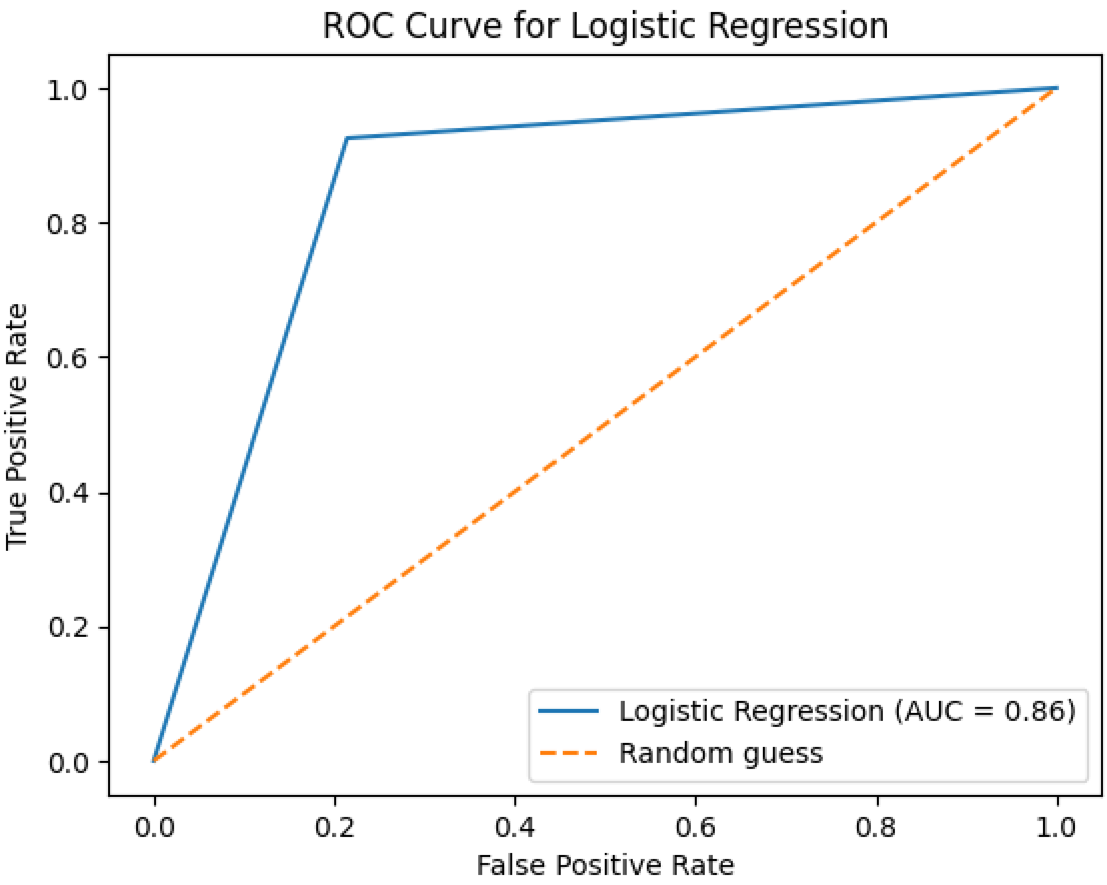
```
Accuracy: 0.8585365853658536
```

In [23]:

```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [24]:

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)
plt.plot(fpr, tpr, label="Logistic Regression (AUC = {:.2f})".format(auc_score))
plt.plot([0, 1], [0, 1], linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend()
plt.show()
```



In [ ]: